WEST

End of Result Set

☐ Generate Collection Print

L10: Entry 1 of 1 File: USPT Oct 5, 1999

DOCUMENT-IDENTIFIER: US 5963939 A
TITLE: Method and apparatus for an incremental editor technology

Detailed Description Paragraph Right (110):
A second set of processes associated with the each object (i.e., question block) in the Question Block Base Class 100 parses the returned answers from the question block, updates the Answers Base Class 140 (FIG. 22B) and then formats and presents the next question block to the client. Preferably, though, only one entry point to the question block is needed to handle all interactions with the client. Each such object possesses three corresponding sets of data driven Question Block processes. The third set of processes then determines, based on responses to previously asked questions, whether to branch to the entry point of other question block objects within the view. The objects of the Questions Blocks do not branch in a tree-like manner nor any other predetermined way, but rather can branch among the other question blocks in a nonlinear, nonsequential manner. Control flow through the question block processes is driven by previous answers. Within the view, the question block identifiers are stored in a queue. Each block of questions has the possibility of invoking one or more other blocks of questions; each block of questions thus invoked is queued for presentation when a previously-queued block of questions is completed. The second set of processes analyzes the Answers Base Class 140 (discussed below) and identifies other question blocks that are to be included.

Detailed Description Paragraph Right (228):
The inter-networking questions relate to the need for firewalls or other security devices among the various network and workgroup 404 topologies, and can be used to identify the need for additional server, hubs 424, routers 426, switches 422, and even cables. Firewalls are mechanisms used in a computer network to prevent, or at least reduce the risk of, unauthorized access to secure data structures. The inter-networking questions are analyzed in the Phase III analysis, culminating in a recommended solution. The recommended solution contains products and inter-networking structures satisfying the detailed product, guidelines, requirements, and questions, corresponding to the Objective Question Blocks 104. However, using the incremental editor, a user of the exemplary embodiment of the present invention can add additional information, by replacing any product in the recommended solution with a product having a slightly higher or slightly lower rank (the concept of rank will be described below), and the user may continue to add additional information via the incremental editor until a final solution with which the user is satisfied is derived.

Detailed Description Paragraph Right (285):
Using information from the Calibration Base Class 280 and from the product object itself, a Search and Match process queries the product database for all products matching the resource requirements. Because each entry in the product database includes a ranking field and a network interconnectability field, the Search and Match process returns a list of appropriate products. The list of products returned by the Search and Match process is placed in a table or subclass, having entry point references that are placed in the objects in the solution framework 400, shown in FIG. 4.

End of Result Set

L6: Entry 1 of 1 File: USPT May 15, 2001

DOCUMENT-IDENTIFIER: US 6233688 B1
TITLE: Remote access firewall traversal URL

Brief Summary Paragraph Right (5):
Specific client software must have support and awareness of specific firewall traversal methods, and thus generic client software cannot be utilized to penetrate the intranet. For example, a client application such as Netscape.TM. may not be able to traverse the firewall since it lacks the means with which to express entry parameters to "support" the private intranet's firewall scheme. Thus, users are often limited to using software that specifically understands and communicates with the intranet. This restricts the choice of client software greatly such that only a limited set of client applications out of all the multitude of programs available can be used when accessing that private intranet.

Detailed Description Paragraph Right (20):
Like FIG. 4a, both the designation of "raft:" and a raft-type are provided. Instead of a traversal point, a generic-URL parameter terminating it. Such an embodiment for the RAFT URL may be used in https or secure http protocols, where the "https" URL designates the entry point through the firewall.

Detailed Description Paragraph Right (28):
For instance, consider the following RAFT URL"
"raft:sslt:https://firewall.foo.com/access.html". This RAFT URL indicates that remote access through the firewall named/addressed as "firewall.foo.com" is to be achieved using secure http by processing the page "access.html" which may be an interactive login page where a user enters login information such as a user name and password for further entry beyond the firewall.

Detailed Description Paragraph Right (32):
Assuming that the RAFT URL is obtained, it is then provided through some interface to the client application (step 520). This interface may be URL data entry dialog of a browser or be chosen from a menu by the user. The RAFT URL may already be provided to the client application by being set-up in a preference manager for the client application or in an operating system registry intended to service that client application. When so provided, the RAFT URL is passed to the socket factory (in Java) (step 530) that will execute methods needed to perform the firewall traversal procedure.

WEST

End of Result Set

Generate Collection Print

L6: Entry 1 of 1 File: USPT May 15, 2001

DOCUMENT-IDENTIFIER: US 6233688 B1
TITLE: Remote access firewall traversal URL

Brief Summary Paragraph Right (5):
Specific client software must have support and awareness of specific firewall traversal methods, and thus generic client software cannot be utilized to penetrate the intranet. For example, a client application such as Netscape.TM. may not be able to traverse the firewall since it lacks the means with which to express entry parameters to "support" the private intranet's firewall scheme. Thus, users are often limited to using software that specifically understands and communicates with the intranet. This restricts the choice of client software greatly such that only a limited set of client applications out of all the multitude of programs available can be used when accessing that private intranet.

Detailed Description Paragraph Right (20):
Like FIG. 4a, both the designation of "raft:" and a raft-type are provided. Instead of a traversal point, a generic-URL parameter terminating it. Such an embodiment for the RAFT URL may be used in https or secure http protocols, where the "https" URL designates the entry point through the firewall.

Detailed Description Paragraph Right (28):
For instance, consider the following RAFT URL"
"raft:sslt:https://firewall.foo.com/access.html". This RAFT URL indicates that remote access through the firewall named/addressed as "firewall.foo.com" is to be achieved using secure http by processing the page "access.html" which may be an interactive login page where a user enters login information such as a user name and password for further entry beyond the firewall.

Detailed Description Paragraph Right (32):
Assuming that the RAFT URL is obtained, it is then provided through some interface to the client application (step 520). This interface may be URL data entry dialog of a browser or be chosen from a menu by the user. The RAFT URL may already be provided to the client application by being set-up in a preference manager for the client application or in an operating system registry intended to service that client application. When so provided, the RAFT URL is passed to the socket factory (in Java) (step 530) that will execute methods needed to perform the firewall traversal procedure.

WEST

End of Result Set

Generate Collection Print

L8: Entry 1 of 1 File: USPT Mar 20, 2001

DOCUMENT-IDENTIFIER: US 6205551 B1
TITLE: Computer security using virus probing

Detailed Description Paragraph Right (5):
As seen from FIG. 1, all communications traffic between public network 100 and private network 130 necessarily passes through firewall 180. In recognition of this communications traffic attribute, I have realized that the firewall 180 provides a preferred location for implementing the security advantages of my invention. Illustratively, in accordance with the preferred embodiment of the invention, firewall 180 illustratively includes processor 181, database 182, and virus prober 185 which randomly inserts probes within incoming files from, e.g., public network 100, to, e.g., private network 130. In accordance with the invention, the probes inserted by the virus prober 185 are individual programs which will trigger particular actions upon execution. In accordance with an embodiment of the invention, the probe is a virus probe configured as a trojan horse which, if executed, on a client will launch a signal back to the firewall indicating that the client is misconfigured. Typically, from a computer virus perspective, a trojan horse is a secret, undocumented entry point placed into a useful application program by an unauthorized user, e.g., computer hacker. In the normal course of execution of the useful application program by a user the trojan horse is also executed thereby launching the undesired actions. Trojan horses are described in more detail in Stallings, supra. at pp. 238-241. For example, a trojan horse can be created to gain access to the files of another user on a shared computer system, wherein the unauthorized user creates a trojan horse program that, when executed, changes the authorized user's file permissions so that their files become readable by any user. This embodiment of the invention utilizes particular features of the trojan horse for delivery of various security advantages to computer networks as discussed in more detail below.

Detailed Description Paragraph Right (9):
Thus, when firewall 180 receives the security alert indication, e.g., UDP packet, that a particular probe has executed (block 210), the firewall will identify the probe and client (block 215) and generate the security alert (block 220.) The nature and type of the security alert generated, in accordance with the invention, can be in a variety of forms. Illustratively, the security alert generated by firewall 180 could be an immediate notification to the network administrator indicating that a particular client or clients within the network currently present a security risk. In a further embodiment of the invention, as probes are executed by various ones of the clients within the network, a log entry is made in a master file, e.g. stored in database 182, which can be accessed by the network administrator at regular intervals or a printed report could be generated from the log for review by the administrator.

L21: Entry 1 of 14 File: USPT Apr 16, 2002

DOCUMENT-IDENTIFIER: US 6373950 B1
TITLE: System, method and article of manufacture for transmitting messages within messages utilizing an extensible, flexible architecture

Brief Summary Paragraph Right (10):
The more well known techniques include magnetic stripe cards purchased for a given amount and from which a prepaid value can be deducted for specific purposes. Upon exhaustion of the economic value, the cards are thrown away. Other examples include memory cards or so called smart cards which are capable of repetitively storing information representing value that is likewise deducted for specific purposes.

Drawing Description Paragraph Right (1):
The foregoing and other objects, aspects and advantages are better understood from the following detailed description of a preferred embodiment of the invention with reference to the drawings, in which:

Detailed Description Paragraph Right (1):
A preferred embodiment of a system in accordance with the present invention is preferably practiced in the context of a personal computer such as the IBM PS/2, Apple Macintosh computer or UNIX based workstation. A representative hardware environment is depicted in FIG. 1A, which illustrates a typical hardware configuration of a workstation in accordance with a preferred embodiment having a central processing unit 10, such as a microprocessor, and a number of other units interconnected via a system bus 12. The workstation shown in FIG. 1A includes a Random Access Memory (RAM) 14, Read Only Memory (ROM) 16, an I/O adapter 18 for connecting peripheral devices such as disk storage units 20 to the bus 12, a user interface adapter 22 for connecting a keyboard 24, a mouse 26, a speaker 28, a microphone 32, and/or other user interface devices such as a touch screen (not shown) to the bus 12, communication adapter 34 for connecting the workstation to a communication network (e.g., a data processing network) and a display adapter 36 for connecting the bus 12 to a display device 38. The workstation typically has resident thereon an operating system such as the Microsoft Windows NT or Windows/95 Operating System (OS), the IBM OS/2 operating system, the MAC OS, or UNIX operating system. Those skilled in the art will appreciate that the present invention may also be implemented on platforms and operating systems other than those mentioned.

Detailed Description Paragraph Right (2):
A preferred embodiment is written using JAVA, C, and the C++ language and utilizes object oriented programming methodology. Object oriented programming (OOP) has become increasingly used to develop complex applications. As OOP moves toward the mainstream of software design and development, various software solutions require adaptation to make use of the benefits of OOP. A need exists for these principles of OOP to be applied to a messaging interface of an electronic messaging system such that a set of OOP classes and objects for the messaging interface can be provided.

Detailed Description Paragraph Right (3):
OOP is a process of developing computer software using objects, including the steps of analyzing the problem, designing the system, and constructing the program. An object is a software package that contains both data and a collection of related structures and procedures. Since it contains both data and a collection of structures and procedures, it can be visualized as a self-sufficient component that does not require other additional structures, procedures or data to perform its specific task. OOP, therefore, views a computer program as a collection of largely autonomous components, called objects, each of which is responsible for a specific task. This concept of packaging data, structures, and procedures together in one component or module is called encapsulation.

Detailed Description Paragraph Right (4):
In general, OOP components are reusable software modules which present an interface that conforms to an object model and which are accessed at run-time through a component integration architecture. A component integration architecture is a set of architecture mechanisms which allow software modules in different process spaces to utilize each others capabilities or functions. This is generally done by assuming a common component object model on which to build the architecture.

Detailed Description Paragraph Right (5):
It is worthwhile to differentiate between an object and a class of objects at this point. An object is a single instance of the class of objects, which is often just called a class. A class of objects can be viewed as a blueprint, from which many objects can be formed.

Detailed Description Paragraph Right (6):
OOP allows the programmer to create an object that is a part of another object. For example, the object representing a piston engine is said to have a composition-relationship with the object representing a piston. In reality, a piston engine comprises a piston, valves and many other components; the fact that a piston is an element of a piston engine can be logically and semantically represented in OOP by two objects.

Detailed Description Paragraph Right (7):
OOP also allows creation of an object that "depends from" another object. If there are two objects, one representing a piston engine and the other representing a piston engine wherein the piston is made of ceramic, then the relationship between the two objects is not that of composition. A ceramic piston engine does not make up a piston engine. Rather it is merely one kind of piston engine that has one more limitation than the piston engine; its piston is made of ceramic. In this case, the object representing the ceramic piston engine is called a derived object, and it inherits all of the aspects of the object representing the piston engine and adds further limitation or detail to it. The object representing the ceramic piston engine "depends from" the object representing the piston engine. The relationship between these objects is called inheritance.

Detailed Description Paragraph Right (8):
When the object or class representing the ceramic piston engine inherits all of the aspects of the objects representing the piston engine, it inherits the thermal characteristics of a standard piston defined in the piston engine class. However, the ceramic piston engine object overrides these ceramic specific thermal characteristics, which are typically different from those associated with a metal piston. It skips over the original and uses new functions related to ceramic pistons. Different kinds of piston engines have different characteristics, but may have the same underlying functions associated with it (e.g., how many pistons in the engine, ignition sequences, lubrication, etc.). To access each of these functions in any piston engine object, a programmer would call the same functions with the same names, but each type of piston engine may have different/overriding implementations of functions behind the same name. This ability to hide different implementations of a function behind the same name is called polymorphism and it greatly simplifies communication among objects.

Detailed Description Paragraph Right (9):
With the concepts of composition-relationship, encapsulation, inheritance and polymorphism, an object can represent just about anything in the real world. In fact, our logical perception of the reality is the only limit on determining the kinds of things that can become objects in object-oriented software. Some typical categories are as follows:

Detailed Description Paragraph Right (10):
With this enormous capability of an object to represent just about any logically separable matters, OOP allows the software developer to design and implement a computer program that is a model of some aspects of reality, whether that reality is a physical entity, a process, a system, or a composition of matter. Since the object can represent anything, the software developer can create an object which can be used as a component in a larger software project in the future.

Detailed Description Paragraph Right (11):
If 90% of a new OOP software program consists of proven, existing components made

from preexisting reusable objects, then only the remaining 10% of the new software project has to be written and tested from scratch. Since 90% already came from an inventory of extensively tested reusable objects, the potential domain from which an error could originate is 10% of the program. As a result, OOP enables software developers to build objects out of other, previously built, objects.

Detailed Description Paragraph Right (12):
This process closely resembles complex machinery being built out of assemblies and sub-assemblies. OOP technology, therefore, makes software engineering more like hardware engineering in that software is built from existing components, which are available to the developer as objects. All this adds up to an improved quality of the software as well as an increased speed of its development.

Detailed Description Paragraph Right (13):
Programming languages are beginning to fully support the OOP principles, such as encapsulation, inheritance, polymorphism, and composition-relationship. With the advent of the C++ language, many commercial software developers have embraced OOP. C++ is an OOP language that offers a fast, machine-executable code. Furthermore, C++ is suitable for both commercial-application and systems-programming projects. For now, C++ appears to be the most popular choice among many OOP programmers, but there is a host of other OOP languages, such as Smalltalk, common lisp object system (CLOS), and Eiffel. Additionally, OOP capabilities are being added to more traditional popular computer programming languages such as Pascal.

Detailed Description Paragraph Right (14):
The benefits of object classes can be summarized, as follows:

Detailed Description Paragraph Right (21):
Thus, as is explained above, a framework basically is a collection of cooperating classes that make up a reusable design solution for a given problem domain. It typically includes objects that provide default behavior (e.g., for menus and windows), and programmers use it by inheriting some of that default behavior and overriding other behavior so that the framework calls application code at the appropriate times.

Detailed Description Paragraph Right (25):
Sun Microsystem's Java language solves many of the client-side problems by:

Detailed Description Paragraph Right (26):
With Java, developers can create robust User Interface (UI) components. Custom "widgets" (e.g. real-time stock tickers, animated icons, etc.) can be created, and client-side performance is improved. Unlike HTML, Java supports the notion of client-side validation, offloading appropriate processing onto the client for improved performance. Dynamic, real-time Web pages can be created. Using the above-mentioned custom UI components, dynamic Web pages can also be created.

Detailed Description Paragraph Right (27):
Sun's Java language has emerged as an industry-recognized language for "programming the Internet." Sun defines Java as: "a simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multithreaded, dynamic, buzzword-compliant, general-purpose programming language. Java supports programming for the Internet in the form of platform-independent Java applets." Java applets are small, specialized applications that comply with Sun's Java Application Programming Interface (API) allowing developers to add "interactive content" to Web documents (e.g. simple animations, page adornments, basic games, etc.). Applets execute within a Java-compatible browser (e.g. Netscape Navigator) by copying code from the server to client. From a language standpoint, Java's core feature set is based on C++. Sun's Java literature states that Java is basically "C++, with extensions from Objective C for more dynamic method resolution".

Detailed Description Paragraph Right (28):
Another technology that provides similar function to JAVA is provided by Microsoft and ActiveX Technologies, to give developers and Web designers wherewithal to build dynamic content for the Internet and personal computers. ActiveX includes tools for developing animation, 3-D virtual reality, video and other multimedia content. The tools use Internet standards, work on multiple platforms, and are being supported by over 100 companies. The group's building blocks are called ActiveX Controls, small, fast components that enable developers to embed parts of software in hypertext markup language (HTML) pages. ActiveX Controls work with a variety of programming

languages including Microsoft Visual C++, Borland Delphi, Microsoft Visual Basic programming system and, in the future, Microsoft's development tool for Java, code named "Jakarta." ActiveX Technologies also includes ActiveX Server Framework, allowing developers to create server applications. One of ordinary skill in the art readily recognizes that ActiveX could be substituted for JAVA without undue experimentation to practice the invention.

Detailed Description Paragraph Right (92):
The table below enumerates the URLs corresponding to the transactions supported by the vPOS Terminal Cartridge. Note that the GET method is allowed for all transactions; however, for transactions that either create or modify information on the merchant server, a GET request returns an HTML page from which the transaction is performed via a POST method.

Detailed Description Paragraph Right (93):
This section describes the GET and POST arguments that are associated with each transaction URL. It also describes the results from the GET and POST methods. For URLs that produce any kind of results, the following fields are present in the HTML document that is returned by the vPOS Terminal Cartridge:

Detailed Description Paragraph Right (94):
For URLs that deal with financial transactions, the following fields are present in the HTML document that is returned by the vPOS terminal cartridge:

Detailed Description Paragraph Right (95):
URL Functionality: This is a local information inquiry function that retrieves the local (merchant's) transaction totals (accumulators).

Detailed Description Paragraph Right (98):
URL Functionality: Corrects the amount of a previously completed transaction.

Detailed Description Paragraph Right (103):
URL Functionality: This transaction is a combination of Auth Only (Authorization without capture) and Forced Post transactions.

Detailed Description Paragraph Right (108):
URL Functionality: Validates the cardholder's account number for a Sale that is performed at a later stage. The transaction does not confirm the sale to the host, and there is no host data capture. The vPOS captures this transaction record and later forwards it to confirm the sale in the Forced Post transaction request.

Detailed Description Paragraph Right (113):
NOTE: The /vPOSt/ci/authonly/ URL should be used for customer-initiated transactions. /vPOSt/mi/authonly/ should be used for merchant-initiated transactions.

Detailed Description Paragraph Right (114):
URL Functionality: Performs an on-line inquiry or the merchant's balance.

Detailed Description Paragraph Right (117):
URL Functionality: Retrieves all records from the transaction log or the batch.

Detailed Description Paragraph Right (120):
URL Functionality: Displays the vPOS terminal configuration data corresponding to the Card Definition Table (CDT).

Detailed Description Paragraph Right (122):
GET Results: The GET method returns a default HTML form that contains the current configuration values. The form can be modified and posted using the /vPOSt/mi/cdt/update/ URL to update the card definition table. Not all fields in the card definition table are editable. The following fields are returned in a form to the user:

Detailed Description Paragraph Right (123):
URL Functionality: Updates the vPOS terminal configuration data corresponding to the Card Definition Table (CDT).

Detailed Description Paragraph Right (125):
GET Results: The GET method returns a default HTML form that contains the current

configuration values. The form can be filled out and posted using the /vPOSt/mi/cdt/update URL to update the card definition table.

Detailed Description Paragraph Right (128):
URL Functionality: Zeroes out the accumulator totals currently resident in the vPOS terminal.

Detailed Description Paragraph Right (133):
URL Functionality: Zeroes out the transaction logs currently batched in the vPOS terminal.

Detailed Description Paragraph Right (138):
URL Functionality: Confirms to the host the completion of a sale, and requests for data capture of the transaction. This is used as a follow-up transaction after doing an Authorization (Online or Off-line) transaction.

Detailed Description Paragraph Right (143):
URL Functionality: Displays the vPOS terminal configuration data corresponding to the Host Definition Table (HDT).

Detailed Description Paragraph Right (145):
GET Results: The GET method returns a default HTML form that contains the current configuration values. The form can be modified and posted using the /vPOSt/mi/hdt/update URL to update the hosts definition table. Not all fields in the host definition table are editable. The following fields are returned in a form to the user:

Detailed Description Paragraph Right (146):
URL Functionality: Updates the vPOS terminal configuration data corresponding to the Host Definition Table (HDT).

Detailed Description Paragraph Right (148):
GET Results: The GET method returns a default HTML form that contains the current configuration values. The form can be filled out and posted to the merchant server using the /vPOSt/mi/hdt/update URL to update the host definition table.

Detailed Description Paragraph Right (149):
URL Functionality: Local function that starts the vPOS at the start of the day.

Detailed Description Paragraph Right (154):
URL Functionality: This transaction is same as the "Authorization Only" transaction, except that the transaction is locally captured by the vPOS terminal without having to communicate with the host. A Forced Post operation is done as a follow-up operation of this transaction.

Detailed Description Paragraph Right (159):
URL Functionality: Downloads the vPOS configuration information from the host and sets up the vPOS in the event of the configuration data being changed.

Detailed Description Paragraph Right (164):
The various configuration parameters can be reviewed and modified using the URLs for the desired functionality.

Detailed Description Paragraph Right (165):
URL Functionality: Used in lodging and hotel establishments to pre-authorize a charge that is completed some time in future.

Detailed Description Paragraph Right (169):
URL Functionality: Completes a pre-authorization transaction.

Detailed Description Paragraph Right (174):
URL Functionality: This transaction is done at the end of the day to confirm to the host to start the settlement process for the transactions captured by the host for that particular vPOS batch.

Detailed Description Paragraph Right (179):
URL Functionality: Credits the return amount electronically to the consumer's account when previously purchased merchandise is returned. The vPOS terminal captures the transaction record for this transaction.

Detailed Description Paragraph Right (184):
URL Functionality: Checks the presence of the host and also the integrity of the link from the vPOS to the host.

Detailed Description Paragraph Right (187):
URL Functionality: This local function locks or stops the vPOS terminal from accepting any transactions.

Detailed Description Paragraph Right (192):
URL Functionality: Cancels a previously completed draft capture transaction.

Detailed Description Paragraph Right (196):
URL Functionality: Administrative transaction used to sign-on the vPOS with the host at the start of the day, and also to download encryption keys for debit transactions.

Detailed Description Paragraph Right (201):
URL Functionality: Returns the vPOS terminal configuration data corresponding to the Communications Parameter Table (CPT).

Detailed Description Paragraph Right (203):
GET Results: The GET method returns a default HTML form that contains the current configuration values corresponding to the vPOS terminal's communication parameters. The form can be filled out and posted to the merchant server using the /vPOSt/mi/cpt/update URL to update the communications parameter table. The following fields are returned in a form to the user:

Detailed Description Paragraph Right (204):
URL Functionality: Updates the vPOS terminal configuration data corresponding to the Communications Parameter Table (CPT).

Detailed Description Paragraph Right (208):
POST Results: On success, the HTML document returned by the vPOS contains the values set by the merchant. On failure, the HTML document contains the reason for the failure of the invocation of the URL.

Detailed Description Paragraph Right (209):
URL Functionality: Returns the vPOS terminal configuration data corresponding to the Terminal Configuration Table (TCT).

Detailed Description Paragraph Right (211):
GET Results: The GET method returns a default HTML form that contains the current configuration values. The form can be filled out and posted using the /vPOSt/mi/tct/update URL to update the terminal configuration table. The following fields are returned in a form to the user:

Detailed Description Paragraph Right (212):
URL Functionality: Updates the vPOS terminal configuration data corresponding to the Terminal Configuration Table (TCT).

Detailed Description Paragraph Right (214):
GET Results: The GET method returns a default HTML form that contains the current configuration values. The form can be filled out and posted using the /vPOSt/mi/tct/update URL to update the terminal configuration table.

Detailed Description Paragraph Right (215):
POST Arguments: All arguments in TCT Review functionality are the returned values from the /vPOSt/mi/tct/update the URL.

Detailed Description Paragraph Right (217):
URL Functionality: Permits the merchant and customer to query a given transaction corresponding to a transaction number.

Detailed Description Paragraph Right (219):
GET Results: For a given transaction, the URL returns an HTML document. If a transaction refers to an older transaction, the transaction's entire history is made available.

Detailed Description Paragraph Right (224):
A brief description of the Virtual Point of Sale Terminal cartridge functions are provided below. vPOSTInit(), vPOSTExec() and vPOSTShut() are the entry points required for each cartridge in accordance with a preferred embodiment. The other functions implement some of the key vPOST cartridge functionality. A source listing of the vPOS code is provided below to further accentuate the detailed disclosure of a preferred embodiment.

Detailed Description Paragraph Right (435):
FIG. 49 shows how the vPOS authenticates an incoming response to a request in accordance with a preferred embodiment. Processing commences at function block 4930 when a message is received by the HTTPS, SET server, or other listener that originated the request to which this reponse corresponds. The message is passed to decision block 4940 to determine if the sending Gateway has transmitted an authentic message and if the gateway is authorized to communicate with this vPOS. If the message is not authentic, then the message is logged as an error or possible attack and the error is handled as shown in function block 4970. If the message is authentic, then the message is decrypted at function block 4950 and the PDU parses the message into name/value pairs. Then, based on the message type and the extended SET version information, the remaining message is parsed at function block 4960 and the message is checked for conformance to the appropriate specification as shown at decision block 4980. If the message does not conform, then it is logged and the error handled at function block 4970. If the message conforms to the proper specification in decision block 4980 then the message is translated into a standardized argument string to be passed to the appropriate executable or code entry point in the vPOS, as shown in function block 4990. Thus, when a vPOS receives an incoming message from a Gateway and parses the Extended SET portion of the message, the message may cause vPOS to execute a program that takes action or queries the user to take action.

Detailed Description Paragraph Right (442):
Using the built-in "serial number" certificate and the Test Gateway public key certificate (which is "hard-wired" into the vPOS sofware), it is possible to securely download a particular bank's customization applications to a specific copy of the vPOS software. Once the vPOS is appropriately configured, the last stage of customization download is to configure the vPOS so that it only responds to a public key certificate of the merchant's acquirer. This process is illustrated here in the context of a merchant who obtains a vPOS that talks to the VeriFone test gateway, and desires to customize the vPOS to interact with a gateway at a bank.

Detailed Description Paragraph Right (455):
The interaction between the vPOS and a client commences when a pay page solicits parameters of a transaction. Then, the parameters are validated to be sure the payment instrument, for example, cardnumber is not null. Then, a transaction object is created, eg. AUTHONLY, and the object is initialized and stuffed with parameters of the transaction, eg. ao.setpan(accnum), and the object is executed. This execution invokes the vPOS engine. The vPOS engine further validates the parameters based on the particular merchant's configuration. For example, some merchans do not accept American Express Cards, but will take Visa, and all merchants check the expiration date of the card. Assuming a valid and acceptable card has been tendered, then a TID is assigned (expiring, existing TIDs) or block a new TID from the TID Pool. This generates a STAN, XID, RRPID unique tag and creates an initial record in the transaction database which is flagged as before gateway processing in case the transaction crashes and must be backed out. Then the protocol parameters are identified in the registry based on card type, and a particular acquirer identified. Then, a protocol object is created and executed to extract results from the protocol object and the before gateway "bit" is flipped to again flag the location of the transaction in the process as it is submitted to the Gateway.

Detailed Description Paragraph Right (456):
The results received back from the Gateway are placed into a transaction object with is reported back to the pay page and ultimatey back to the pay page user.

Detailed Description Paragraph Right (457):
A novel feature of the vPOS software provides payment page customization based on a merchant's preferences. This feature automatically lists cards that are accepted by a particular merchant based on the active terminal configuration. Each approved card for a particular merchant is linked to the display via an URL that provides a pointer to the credit card information supported by the merchant. Each card has an

entry in a data structure referred to as the Card Definition Table (CDT).

Detailed Description Paragraph Right (458):
A preferred embodiment of the vPOS merchant pay customization software in accordance with a preferred embodiment is provided in FIG. 19 which illustrates the logic utilizing a flowchart, and a listing of the source code below. Processing commences at terminal 1900 and immediately flows to function block 1910 where an index variable is initialized for stepping through each of the accepted payment instruments for the merchant's page. Then, at function block 1930, a URL key is obtained associated with the current merchant pay page and index value. The URL key is a registry key name that points to a picture of a credit card that the merchant has associated with the pay page and which the merchant accepts as payment. At output block 1940 the card image associated with the URL key is obtained and displayed on the terminal. The CDT entry is obtained at function block 1950 utilizing the URL key. The CDT is utilized for storing information associated with each card. Then, at decision block 1960, a test is performed to determine if the last payment method card has been processed and displayed on the merchant display. If not, then the index is incremented at function block 1920 and the loop reiterated to process the next card at function block 1930. If all the cards have been processed, then control is returned to the merchant program for processing the transaction at terminal 1970.

Detailed Description Paragraph Right (479):
A virtual, private network between the Gateway and the host processor is established to expedite host communication. In addition, two Network Interface Processors (NIP)s--a "near end" NIP that interfaces to the Gateway and a "far end" NIP that interfaces to the host. The NIPs will handle virtual connections between themselves. The far-end NIP will take care of specific communication details. The near-end NIP is an IP-addressable device that converts between TCP messages and packets. It is installed on a public network 2330, which is a LAN outside the corporate firewall. The Gateway, on the secure public network 2330, utilizes TCP/IP 2320 to communicate with the near-end NIP.

Detailed Description Paragraph Right (482):
At the application level, SET provides signed and encrypted data encapsulations of payment information portions of the transaction messages. Transport-level encryption of the entire message packet is required for additional security. The HTTPS protocol--i.e., HTTP over SSL 3.0--is utilized between the merchants and the Gateway. The virtual connections between the near-end NIP and the host are part of a private network. The termination will occur outside the firewall. Data between the Gateway and the host is sent in the clear with no encryption. In this network configuration, a transaction between a merchant's vPOS and the host will cross the firewall four times: SET request from vPOS to Gateway, legacy request from Gateway to NIP, LEGACY response from NIP back to Gateway, and SET response from Gateway back to vPOS.

Detailed Description Paragraph Right (557):
The Payment Manager 2606 coordinates and completes the payment request that is received from the merchant system. The payment request is received via a MIME message in the native code implementation or via an applet in the Java implementation. The payment request received contains the final GSO, Ship-To name, merchant certificate, merchant URL, coupons and the payment amount. The manager 2606 then communicates with the payment related GUI component to interact with the consumer to authorize and complete the payment transaction. The manager is also responsible for determining the payment protocol based on the consumer's payment instrument and the merchant's preferred payment protocol.

Detailed Description Paragraph Right (560):
The payment manager 2730 also sends and receives transactions to the protocol manager 2770 including a merchant's payment message 2760, a consumer certificate and PK handle 2750, a merchant URL 2742, a payment 2740, a signed receipt 2734 and a GSO, Selected Payment Protocol and Selected Payment Instrument 2732. The payment manager 2730 also accepts input from the payment applet or MIME message from the merchant as shown at function block 2780. One aspect of the payment processing is a Consumer Payments Class Library (CPCL) 2770 which encapsulates the payment protocols into a single API. By encapsulating the payment protocols, applications are insulated from protocol variations. A SET Protocol provides an implementation of the client-side component of the Secure Electronic Transaction (SET) Protocol. A complete implementation of the client-side component of the CyberCash micro-payment

protocol is also provided.

Detailed Description Paragraph Right (567):
A Data Manager provides storage and retrieval of generic data items and database records. It is assumed that data fields, index fields or entire data records can be marked as encrypted and the encryption process is largely automated. The data manager has no specific knowledge of database records appropriate to different payment methods. This layer is separated out so as to reduce changes required when new payment methods are introduced. However RSA key pairs and certificates might be considered as "simple" data types. This component also provides an abstraction which supports wallet files on computer disk or contained in smart cards.

Detailed Description Paragraph Right (568):
The Open Data Base Connectivity (ODBC)/Java Data Base Connectivity (JDBC) component provides Data Base Connectivity where formal database components are required. An embodiment of the Smart Card Wallet allows wallet data to be stored and/or secured by a cryptographic token.

Detailed Description Paragraph Right (569):
A preferred embodiment includes a single file or directory of files comprising a "wallet" which contains personal information and information about multiple payment methods with the preferred implementation. These payment methods (Visa cards, debit cards, smart cards, micro-payments etc.) also contain information such as account numbers, certificates, key pairs, expiration dates etc. The wallet is envisaged to also contain all the receipts and transaction records pertaining to every payment made using the wallet. A Cryptographic API component provides a standard interface for RSA and related cryptographic software or hardware. This support includes encryption, signature, and key generation. Choice of key exchange algorithm, symmetric encryption algorithm, and signature algorithm should all be configurable. A base class stipulates generic behavior, derived classes handle various semantic options (e.g. software based cryptography versus hardware based cryptography.)

Detailed Description Paragraph Right (571):
FIG. 28 is a Consumer Payment Message Sequence Diagram in accordance with a preferred embodiment of the invention. The diagram presents the flow of messages between the consumer, the browser, the merchant system, the PayWindow application, and CPCL. This message flow describes the payment process from the time an order is completed and the consumer elects to pay, to the time the payment is approved and the receipt is returned to the consumer. The difference between the Native implementation and Java implementation of the PayWindow application is in the delivery of the order information to the PayWindow. Once the order information is received by the PayWindow, the flow of messages/data is the same for both implementations. In the case of the Native implementation, the order information is delivered via a MIME message. This MIME message is sent to the PayWindow by the browser via a document file. In the Java implementation, the order information is delivered to the PayWindow by an applet. The merchant system sends an applet with the order information to the browser which in turn delivers the order to the PayWindow. Once the order is received, the PayWindow interacts with the consumer and the Protocol modules for the completion of the payment process.

Detailed Description Paragraph Right (574):
On receipt of the order, the merchant system calculates the payment amount. This message represents the HTML page which is sent by the merchant system detailing the payment amount along with the Java payment applet which contains the GSO, PPPs, AIs, merchant certificate and URL.

Detailed Description Paragraph Right (575):
The Java enabled browser runs the Payment applet. The applet displays a button called "Pay" for the consumer to click. This is embedded in the HTML page delivered by the merchant.

Detailed Description Paragraph Right (577):
This message represents the GSO, PPPs, AMs, merchant certificate and the merchant URL carried by the Java applet. The Java applet now delivers these to the PayWindow application.

Detailed Description Paragraph Right (584):
These messages represent the merchant's URL, the GSO, payment protocol (PP) to use, account number, certificate and the private key handle (PK) associated with the

payment instrument which is sent to the protocol module.

Detailed Description Paragraph Right (594):
FIG. 33 illustrates a selected payment instrument with a fill in the blanks for the cardholder in accordance with a preferred embodiment. Next time the payment instrument holder is opened in a payment context the certificate issuing authorty's approved instrument bitmap can be used to select the payment instrument and utilize it to make purchases. FIG. 34 illustrates a coffee purchase utilizing the newly defined VISA card in accordance with a preferred embodiment of the invention.

Detailed Description Paragraph Right (604):
FIG. 54 depicts the step of host communication, as shown in Step 5145 in FIG. 51. Execution begins in Step 5400. In Step 5405 the Gateway obtains from the request object the string representing the request text. In Step 5410 it obtains the sequence number for the request. In Step 5415 the Gateway determines the current time, in order to record the time at which the request is made. In Step 5420 the Gateway sends the request to the host and waits for a response from the host. When a response is received, execution continues in Step 5425. In Step 5425, the Gateway again checks the current time, thereby determining the time at which a response was received. In Step 5430, the Gateway checks to see whether the communication was successfully performed. If a communication was not successful, the Gateway records that an error occurred in Step 5432. If the communication was successful, the Gateway, in Step 5435, indicates that the request was successfully sent and responded to. In Step 5437, the Gateway sets the response string based upon the response received in Step 5420. In Step 5439 the Gateway sets a status to indicate that reverse translation of the received response is required. Regardless of whether the communication was successful or unsuccessful, execution continues to Step 5450. In Step 5450, the database is updated with status information from the host communication. In Step 5490, control is returned to the calling routine.

Detailed Description Paragraph Right (606):
FIG. 55 depicts the operation of the TranslateReverse routine, as executed in Step 5155 in FIG. 51. Execution begins at Step 5500. In Step 5510 the Gateway reverse-translates the response received from the legacy system host. Reverse translation consists of extracting data from the data records received from the legacy system, and placing them in objects so that they are useable by the Gateway. In Step 5520, the Gateway checks to verify that translation was successful. If translation was successful control proceeds to Step 5530, where a status flag is set indicating a successful translation. If translation was not successful, control proceeds to Step 5540, in which the Status Flag is set to indicate an unsuccessful translation. Regardless of whether translation was successful or unsuccessful, execution proceeds to Step 5550. In Step 5550, a status flag is set to indicate that the next stage for the request is to provide a response from the Gateway. This step is always executed, because, regardless of whether the translation or any other aspect of the transaction was successful, a response indicating either success or failure must be returned by the Gateway. Control then proceeds to Step 5590, in which the TranslateReverse routine returns control to the calling routine in FIG. 51. It will be seen that the TranslateForward routine in FIG. 53, the DoHostCommunication routine depicted in FIG. 54, and the TranslateReverse routine depicted in FIG. 55, each alter the status of the request. As a result as the loop depicted in FIG. 51 executes a particular request will proceed through all three stages and finally to exit in Step 5190.

Detailed Description Paragraph Right (608):
FIGS. 56A and 56B describe the GetSetKeyFields routine. This routine is called by Step 5205 as illustrated in FIG. 52A. Execution begins in Step 5600. In Step 5610, the Gateway interrogates the request object to determine the request type. In Step 5620, the Gateway determines whether the request type is for authorization only. If the request type is not for authorization only, execution proceeds to Step 5625. In Step 5625, the Gateway checks to see whether the request type is for a sale. If the request type is neither for authorization only nor for a sale, execution proceeds to Step 5630. In Step 5360, the Gateway indicates that the request type is not a supported request, and proceeds to Step 5635, where it returns to the caller.

Detailed Description Paragraph Right (609):
If the request type is either for authorization only or for a sale, execution proceeds with Step 5640. In step 5640, the Gateway initializes a container object to represent the request. In Step 5650, the Gateway extracts the [transaction identifier?] (XID) for the transaction. In Step 5652, the Gateway extracts the

merchant identifier (MID) for the transaction. In Step 5654, the Gateway extracts the [what is the RRPID?] (RRPID) and the terminal identifier (TID) for the request. In Step 5656, the Gateway extracts the retry count associated with the current request. In Step 5660, a message data area is initialized with the extracted contents. The message area can then be used for further processing by the called routine. In Step 5690, the GetSetKeyFields routine returns control to the caller.

Detailed Description Paragraph Left (116):
Payment Applet with GSO, PPPs, AIs, Merchant Certificate and URL 2840

Detailed Description Paragraph Left (119):
GSO, PPPs, AMs, merchant certificate and URL 2860

Detailed Description Paragraph Center (16):
URL Table

Detailed Description Paragraph Center (17):
URL Descriptions

Detailed Description Paragraph Center (47):
URL Results

Detailed Description Paragraph Type 1 (1):
Objects can represent physical objects, such as automobiles in a traffic-flow simulation, electrical components in a circuit-design program, countries in an economics model, or aircraft in an air-traffic-control system.

Detailed Description Paragraph Type 1 (2):
Objects can represent elements of the computer-user environment such as windows, menus or graphics objects.

Detailed Description Paragraph Type 1 (3):
An object can represent an inventory, such as a personnel file or a table of the latitudes and longitudes of cities.

Detailed Description Paragraph Type 1 (4):
An object can represent user-defined data types such as time, angles, and complex numbers, or points on the plane.

Detailed Description Paragraph Type 1 (5):
Objects and their corresponding classes break down complex programming problems into many smaller, simpler problems.

Detailed Description Paragraph Type 1 (6):
Encapsulation enforces data abstraction through the organization of data into small, independent objects that can communicate with each other. Encapsulation protects the data in an object from accidental damage, but allows other objects to interact with that data by calling the object's member functions and structures.

Detailed Description Paragraph Type 1 (7):
Subclassing and inheritance make it possible to extend and modify objects through deriving new kinds of objects from the standard classes available in the system. Thus, new capabilities are created without having to start from scratch.

Detailed Description Paragraph Type 1 (8):
Polymorphism and multiple inheritance make it possible for different programmers to mix and match characteristics of many different classes and create specialized objects that can still work with related objects in predictable ways.

Detailed Description Paragraph Type 1 (9):
Class hierarchies and containment hierarchies provide a flexible mechanism for modeling real-world objects and the relationships among them.

Detailed Description Paragraph Type 1 (12):
Call versus override. With a class library, the code the programmer instantiates objects and calls their member functions. It's possible to instantiate and call objects in the same way with a framework (i.e., to treat the framework as a class library), but to take full advantage of a framework's reusable design, a programmer typically writes code that overrides and is called by the framework. The framework

manages the flow of control among its objects. Writing a program involves dividing responsibilities among the various pieces of software that are called by the framework rather than specifying how the different pieces should work together.

Detailed Description Paragraph Type 1 (33):
Smart Cards

Detailed Description Paragraph Type 1 (367):
// Creates the Transaction Object, takes care

Detailed Description Paragraph Type 2 (2):
Flow of control. A program written with the aid of class libraries is still responsible for the flow of control (i.e., it must control the interactions among all the objects created from a particular library). The programmer has to decide which functions to call at what times for which kinds of objects.

Detailed Description Paragraph Table (1):
Transaction URL POST Access Control HOST FINANCIAL PAYMENT FUNCTIONALITY auth capture /vPOSt/mi/authcapture/ allowed merchant login/password auth capture /vPOSt/ci/authcapture/ allowed no access control auth only /vPOSt/mi/authonly/ allowed merchant login/password auth only /vPOSt/ci/authonly/ allowed no access control adjust /vPOSt/mi/adjust/ allowed merchant login/password forced post /vPOSt/mi/forcedpost/ allowed merchant login/password offline auth /vPOSt/mi/offlineauth/ allowed merchant login/password offline auth /vPOSt/ci/offlineauth/ allowed no access control pre auth /vPOSt/mi/preauth/ allowed merchant login/password pre auth /vPOSt/mi/preauthcomp/ allowed merchant comp login/password return /vPOSt/mi/return allowed merchant login/password return /vPOSt/ci/return/ allowed no access control void /vPOSt/mi/void/ allowed merchant login/password HOST ADMINISTRATIVE PAYMENT FUNCTIONALITY balance /vPOSt/mi/bi/ not allowed merchant inquiry login/password host logon /vPOSt/mi/hostlogon/ allowed merchant login/password parameter /vPOSt/mi/parameters not allowed merchant download dnld/ login/password reconcile /vPOSt/mi/reconcile/ allowed merchant login/password test host /vPOSt/mi/testhost/ not allowed merchant login/password LOCAL FUNCTIONS & TRANSACTIONS accum /vPOSt/mi/accum/review/ not allowed merchant review login/password batch review /vPOSt/mi/batch/review/ not allowed merchant login/password cdt review /vPOSt/mi/cdt/review/ not allowed merchant login/password cdt update /vPOSt/mi/cdt/update/ allowed merchant login/password cpt review /vPOSt/mi/cpt/review not allowed merchant login/password cpt update /vPOSt/mi/cpt/update/ allowed merchant login/password clear accum /vPOSt/accum/clear/ allowed merchant login/password clear batch /vPOSt/mi/batch/clear/ allowed merchant login/password hdt review /vPOSt/mi/hdt/review/ not allowed merchant login/password hdt update /vPOSt/mi/hdt/update/ allowed merchant login/password lock vPOS /vPOSt/mi/lock/ allowed merchant login/password query txn /vPOSt/ci/querytxn/ not allowed no access control query txn /vPOSt/mi/querytxn/ not allowed merchant login/password tct review /vPOSt/mi/tct/review/ not allowed merchant login/password tct update /vPOSt/mi/tct/update/ allowed merchant login/password unlock vPOS /vPOSt/mi/unlock/ allowed merchant login/password

Detailed Description Paragraph Table (22):
vPOSTInit() /* vPOST cartridge Initialization here */ WRBReturnCode vPOSTInit(void **clientCtx){ vPOSTCtx *vPOSTCxp ; /* Allocate memory for the client context */ if (!(vPOSTCxp = (vPOSTCtx *)malloc(sizeof(vPOSTCtx)))) return WRB_ERROR ; *clientCtx = (void *)vPOSTCxp ; return (WRB_DONE) ;} vPOSTShut() WRBReturnCode vPOSTShut{ void *WRBCtx, void *clientCtx){ *WRBCtx ; /* not used */ assert(clientCtx) ; /* Free the client context allocated in vPOSTInit() routine free(clientCtx) ; return (WRB_DONE) ;} vPOSTExec() /* The driver cartridge routine */ WRBReturnCode vPOSTExec(void *WRBCtx, void *clientCtx) { vPOSTCtx *vPOSTCxp ; char *uri ; char *txnMethod ; /* HTTP method */ enum evPOSTTxn *txn ; /* vPOST transaction */ char *txnOutFile ; /* Output file from transaction */ char **txnEnv ; /* environment variables values for transaction */ char *txnContent ; /* transaction's POST data content */ WRBEntry *WRBEntries ; int numEntries ; vPOSTCxp = (vPOSTCtx *) clientCtx ; /* WRBGetURL gets the URL for the current request */ if (!(uri = WRBGetURL(WRBCtx))) return (WRB_ERROR) ; /* WRBGetContent() gets theQueryString/POST data content */ if (!(txnContent = WRBGetContent(WRBCtx))) { return WRB_ERROR ; } /* WRBGetParserContent() gets the parsed content */ if (WRB_ERROR == WRBGEtParsedContent(WRBCtx, &WRBEntries, &numEntries)) { return WRB_ERROR ; } /* WRBGetEnvironment() gets the HTTP Server Environment */ if (!(txnEnv = WRBGetEnvironment(WRBCtx))) { return WRB_ERROR ; } /* vPOSTGetMethod() gets the

method for the current request */ if (!(method = vPOSTGetMethod(txnEnv))){ return (WRB_ERROR) ; } /* vPOSTGetTxn() gets the vPOST transaction for the request */ txn = vPOSTGetTxn(uri) ; if (eTxnError == txn) { return (WRB_ERROR) ; } /* vPOSTExecuteTransaction() executes the vPOST transaction */ txnOutFile = vPOSTExecuteTransaction(WRBCtx, txn, txnMethod, txnEnv, txnContent) ; if (!(txnOutFile)) { return (WRB_ERROR) ; } /* Write out the file */ vPOSTWriteFile(txnOutFile) ; return (WRB_DONE) ; } vPOSTGetTxn() enum evPOSTTxn vPOSTGetTxn(char *uri) { /* * The function scans the uri and extracts the string * corresponding to the transaction and returns it to the * caller */ }

Detailed Description Paragraph Table (33):
typedef struct vPOSParamsBlk { char szTransAmt[ ]; // Without decimal point. // Left most two digits implied to be decimal digits char szPurchOrderNum[ ]; char szRetRefNum[ ]; char szBatchNum[ ]; char szNewBatchNum[ ]; char szOrigAmt[ ]; char szCPSData[ ]; char szAuthId[ ]; // Auth Id for offline auth-only transaction int HostIndex; unsigned int nTransRefNum; vPOSBool fvPOSLock; ECPSDataType eCPSType; EPCLTransType TransType; EStatus TransResult; EPCLPmtInst PmtInst; EPCLCurrency CurrencyType; EPCLDecimals NumDecDigits; EVPCLAccumType AccumType; TPmtInstData PayInstData; union vPOSConfigData { TvPOSHDTRec srHDTRec; TvPOSCDTRec srCDTRec; TvPOSCPTRec srCPTRec; TvPOSTCTRec srTCTRec; } vPOSConfigData; void *Context; // Context from the calling interface EStatus (*vPOSCallBack)(TvPOSResultsBlk *, void *); } TvPOSParamsBlk;

Detailed Description Paragraph Table (35):
#include "rr.h" #ifndef NT #define NT extern void setenvp( ); #endif // // AcquireBillHtml // On Pay page, output form entries to acquire billing information // EStatus AcquireBillHtml(CWSINT& clWSINT, int nTot, CProf& clProfile, EPCLCurrency eCurrency) { // Current time time_t tNow; //figure out current year for Credit card expiry struct tm *tmNow; char szYear[DB_YEAR_SZ + 1]; char szAmount[FORMATTED_CURRENCY + 1]; time(&tNow); tmNow = localtime(&tNow); strftime(&szYear[0], (size_t)DB_YEAR_SZ + 1, "%Y", tmNow); // needs extra 1 for null int nYear = atoi(szYear); /*<TH>Payment Type</TH>.backslash.n<TD><INPUT SIZE = 20 NAME=b_instrument VALUE=.backslash."".backslash. <<clProfile.m_b_instrument << ".backslash."></TD>".backslash. << "*/ clWSINT << "<CENTER><TABLE BORDER=0><CAPTION ALIGN = TOP><B>Bill To</B></CAPTION>.backslash.n"; clWSINT << "<TR ALIGN=LEFT ><TH>Account Number</TH><TD COLSPAN = 5><INPUT SIZE = 56 MAXLENGTH = " << ACCT_NUM_SZ << "NAME=b_card> </TD></TR>.backslash.n"; clWSINT << "<TR ALIGN=LEFT><TH>Name on Card</TH><TD><INPUT SIZE= 20 MAXLENGTH= "<< NAME_SZ << "NAME=b_name VALUE=.backslash."" <<clProfile.m_b_name << ".backslash."> </TD><TH>Expiration</TH><TD>Month <SELECT NAME = b_expire_month><OPTION> 01.backslash.n <OPTION> 02.backslash.n" << "<OPTION> 03.backslash.n <OPTION> 04.backslash.n<OPTION> 05.backslash.n<OPTION> 06.backslash.n<OPTION> 07.backslash.n<OPTION> 08 .backslash.n<OPTION> 09 .backslash.n" << "<OPTION> 10.backslash.n<OPTION> 11.backslash.n<OPTION> 12.backslash.n</SELECT> Year <SELECT NAME = b_expire_year><OPTION>" << nYear << "<OPTION>" << nYear + 1 << "<OPTION>" << nYear + 2 << "<OPTION>" <<nYear + 3 << "<OPTION>" << nYear + 4 << "</SELECT></TD></TR>.backslash.n"; //<TH>Expires</TH><TD>Month <INPUT SIZE=3 NAME=b_expire_month> Year <INPUT SIZE=5 NAME=b_expire_year></TD></TR>.backslash.n"; clWSINT << "<TR ALIGN=LEFT><TH>Address Line 1</TR><TD COLSPAN=5><INPUT SIZE=56 MAXLENGTH= " << ADDR_SZ << " NAME=b_addr1 VALUE=.backslash."" << clProfile.m_b_addr1 << ".backslash."> </TD></TR>.backslash.n"; clWSINT << "<TR ALIGN=LEFT><TH>Address Line 2</TH><TD COLSPAN=5><INPUT SIZE=56 MAXLENGTH= " << ADDR_SZ << " NAME=b_addr2 VALUE=.backslash."" << clProfile.m_b_addr2 << ".backslash."></TD></TR>.backslash.n"; clWSINT << "<TR ALIGN=LEFT><TH>City</TH><TD><INPUT MAXLENGTH= " << CITY_SZ << "NAME=b_city VALUE=.backslash."" << clProfile.m_b_city << ".backslash."> </TD>" << "<TH>State/Province</TH><TD><INPUT MAXLENGTH= " << STATE_SZ << " NAME=b_state VALUE=.backslash."" << clProfile.m_b_state << ".backslash."> </TD></TR >.backslash.n"; clWSINT << "<TR ALIGN=LEFT><TH>Country</TH><TD><INPUT MAXLENGTH= " << COUNTRY_SZ << " NAME=b_country VALUE=.backslash."" <<clProfile.m_b_country << ".backslash."> </TD><TH>Zip/Postal Code</TH><TD><INPUT MAXLENGTH= " << ZIP_SZ << " NAME=b_zip VALUE=.backslash."" << clProfile.m_b_zip << ".backslash."> </TD></TR>.backslash.n"; clWSINT << "<TR ALIGN=LEFT><TH>Email</TH><TD><INPUT MAXLENGTH= " >> BEMAIL_SZ << " NAME=b_email VALUE=.backslash."" << clProfile.m_b_email << " "> <.backslash.TD>" << "<TH>Phone</TH><TD><INPUT MAXLENGTH= " << BPHONE_NUM_SZ << " NAME=b_phone VALUE=.backslash."" << clProfile.m_b_phone << ".backslash."> </TD></TR>.backslash.n"; clWSINT << "</TABLE ></CENTER><P>.backslash.n"; //NPW<< " NAME=b_addr1> </TD>" << "<TH>Payment

```
Instrument</TH>.backslash.n<TD><SELECT NAME =b_instrument>"; //hack from ini (bug)
which pay instruments supported //NPW clWSINT << "<OPTION> Credit Card.backslash.n"
<< "<OPTION> Debit Card.backslash.n</SELECT></TD></TR>.backslash.n";
CurrFormat(nTot, eCurrency, szAmount); clWSINT << "<CENTER><FONT SIZE=5>Total =" <<
szAmount<< "</FONT></CENTER>"; return (eSuccess); }
////////////////////////////////////////////////////////////// // PayButtonsHtml //
Output buttons on pay page: return to shop, pay, pay window, // modify order
////////////////////////////////////////////////////////////// void
PayButtonsHtml(CWSINT& cIWSINT, char* pszShopUrl, CRRReg& clReg) { char *pszHomeUrl
= clWSINT.LookUp("home_url"); char *pszModifyUrl = clWSINT.LookUp("modify_url");
char *pszSoftUrl = clWSINT.LookUp("soft_url"); if (!pszHomeUrl) pszHomeUrl =
pszShopUrl; //Home Page //if (!pszModifyUrl) pszModifyUrl = pszShopUrl; //Shopping
Cart typically clWSINT << "<CENTER><H4>By pressing the Pay! button I agree to pay
the above total amount<br> according to the card issuer
agreement<H4></CENTER>.backslash.n"; clWSINT << "<CENTER>.backslash.n<A HREF = " <<
pszShopUrl << "> <IMG SRC=" << clReg.m szReturnShop << "BORDER =
0></A>.backslash.n"; #ifdef_SC clWSINT << "<INPUT TYPE = IMAGE NAME = gso SRC = " <<
clReg.m_szModifyOrder << "BORDER = 0>.backslash.n"; #else if (pszModifyUrl) clWSINT
<< "<A HREF = " << pszModifyUrl << "> <IMG SRC=" << clReg.m_szModifyOrder << "
BORDER = 0></A>.backslash.n"; #endif clWSINT << "<INPUT TYPE = HIDDEN NAME =
home_url VALUE = " << pszHomeUrl << ">.backslash.n" << "<INPUT TYPE = IMAGE NAME =
vPOS SRC = " << clReg.m_szPay << " BORDER = 0>.backslash.n" << "<INPUT TYPE = HIDDEN
NAME = shop_url VALUE = " << pszShopUrl << ">.backslash.n" << "<INPUT TYPE = HIDDEN
NAME = store VALUE = " << clWSINT.LookUp("store") << ">.backslash.n";//Can't be NULL
or error previously if(pszSoftUrl) clWSINT << "<INPUT TYPE = HIDDEN NAME = soft_url
VALUE = " << pszSoftUrl << ">.backslash.n"; clWSINT << "</CENTER>.backslash.n"; }
////////////////////////////////////////////////////////////// // DisplayPayPage //
Outputs billing form, buttons, and static gso
////////////////////////////////////////////////////////////// EStatus
DisplayPayPage(CWSINT& clWSINT, CRRReg& clReg, int nError) { EStatus eStat; char
szFileLine[BUFFER_SZ + 1]; char *pszTag, *pszRefererUrl, *pszShopUrl, *pszExePath,
*pszServerName; time_t tNow; int nTagExist = FALSE; HKEY hCardsKey; //To enumerate
cards long retCode; int nNoCards; DWORD dwtype, dwlen; HKEY hCardKey; char
szCardBuf[MAX_PATH + 1], szCardPic[MAX_PATH + 1]; #ifdef_SC CPOLBk clBkGso; #else
char *pszTxn, *pszGsoNum, *pszGsoOpaque, *pszTot; #endif // Shipping headers. If
come from gso page and cookies are not set, set. CProf *pProfile; pProfile = new
CProf( ); if (!pProfile) return (eRRNewFailed); eStat = pProfile->Init(clWSINT);
if(eStat != eSuccess) return (eStat); //Init failed #ifdef_SC /*NO session cookie
for the pay page. This means the user will either use a long term cookie or type in
their info each time*/ clWSINT << "Set-Cookie: profile=" << pProfile->GetCookieLine(
) << "; path=/.backslash.n"; /* if(clWSINT.LookUp("Server Name")) clWSINT << ";
domain = "<< clWSINT.LookUp("Server Name") << ";.backslash.n";*/ #endif #ifdef_SC //
Shipping filled in? if (!(pProfile->m_s_name[0] && pProfile->m_s_addr1[0] &&
pProfile->m_s_city[0] && pProfile->m_s_state[0] && pProfile->m_s_zip[0] &&
pProfile->m_s_country[0] && pProfile->m_s_ship[0])) { eStat =
DisplayGsoPage(clWSINT, clReg, ERROR_DISPLAY); //bug, return correct? return eStat;
} // Creates shopping basket from CGI/Cookies eStat = clBkGso.Init(clWSINT,
*pProfile, clReg); if (eStat != eSuccess) return (eStat); //eRRBasketCreateError //
Cookies then other headers clBkGso.ToCookies(clWSINT, REGULAR); #endif // clWSINT <<
"Pragma: no-cache .backslash.n"; clWSINT << "Content-type:
text/html.backslash.n.backslash.n"; //Where to position the page. if all information
is filled in, here. if(!nError) {clWSINT << "<A NAME=jump></A>";} //Output HTML
ifstream ifPay; ifPay.open(clReg.m_szPayTemplate, ios::in.vertline.ios::nocreate);
if (ifPay.fail( )) return (eRRcantOpenPayTemplate); //couldn't read pay template
file // HTML Template while (ifPay) { ifPay.getline(szFileLine, BUFFER_SZ); if
(!(pszTag =strstr(szFileLine, DYNAMIC_TAG))) clWSINT << szFileLine <<
".backslash.n"; else { nTagExist = TRUE; // Null the tag, Output the beginning of
the line, //make the dynamic basket call, output the rest of the line if
(strlen(szFileLine) == strlen(DYNAMIC_TAG)) pszTag[0] = NULL; else { pszTag[0] =
(char) NULL; pszTag += strlen(DYNAMIC_TAG) + 1; //was 9 }

Detailed Description Paragraph Table (36):
clWSINT << szFileLine; // Dynamic call pszRefererUrl = clWSINT.LookUp("Referer"); if
(!pszRefererUrl) return (eRRNoRefererUrl); pszExePath = clWSINT.LookUp("Executable
Path"); if (!pszExePath) return (eRRNoExePath); pszServerName =
clWSINT.LookUp("Server Name"); if (!pszServerName) return (eRRNoServerName); clWSINT
<< "<FORM METHOD = POST ACTION = http"; if (clReg.m_nUseSSL) clWSINT << "s"; clWSINT
<< "://" <<pszServerName << pszExePath << "#jump>"; /*clWSINT << "<FORM METHOD =
POST ACTION =" << pszExePath << "#jump>";*/ // Setting Long Cookies clWSINT <<
```

```
"<CENTER >If you wish to have billing and shipping defaults set in your browser,
check this box" << "<INPUT TYPE = CHECKBOX NAME=long_cookies><CENTER>.backslash.n";
//Fill it in message if (nError) { clWSINT << "<A NAME=jump></A>"; clWSINT <<
"<CENTER><H4>You must fill in <I>all</I>of the billing information except for
<I>address line 2</I>and <I>email</I>.</H4></CENTER >"; } //GsoNum #ifdef_SC
time(&tNow); // For multithreading, append instantiation number clWSINT << "<TABLE
ALIGN=RIGHT><TR><TH>Order Number</TH><TD>" << tNow << "</TD></TR></TABLE><BR
CLEAR=ALL>.backslash.n<INPUT TYPE=HIDDEN NAME=b_gso_num VALUE = " << tNow <<
">.backslash.n"; #else //Pay page API: transaction type, GSO #, gso opaque pszGsoNum
= clWSINT.LookUp("b_gso_num"); if (pszGsoNum) clWSINT << "<TABLE
ALIGN=RIGHT><TR><TH>Order Number</TH><TD>" << pszGsoNum << "</TD></TR></TABLE><BR
CLEAR=ALL>.backslash.n<INPUT TYPE=HIDDEN NAME=b_gso_num VALUE = " << pszGsoNum <<
">.backslash.n"; else{ time(&tNow); //For multithreading, append instantiation
number clWSINT << "<TABLE ALIGN=RIGHT><TR><TH>Order Number</TH><TD>" << tNow <<
"</TD></TR></TABLE><BR CLEAR=ALL>.backslash.n<INPUT TYPE=HIDDEN NAME=b_gso_num VALUE
=" << tNow << ">.backslash.n"; } // Some pay page only specifics: transaction to
execute, gso opaque pszTxn = clWSINT.LookUp("transaction"); if(pszTxn) clWSINT <<
"<INPUT TYPE=HIDDEN NAME=transaction VALUE = " << pszTxn << ">.backslash.n";
pszGsoOpaque = clWSINT.LookUp("gso_opaque"); if (pszGsoOpaque) clWSINT << "<INPUT
TYPE=HIDDEN NAME=gso_opaque VALUE = .backslash."" << pszGsoOpaque <<
".backslash.".backslash.">.backslash.n"; #endif #ifdef_SC // Bill to information &
Payment Instrument eStat = AcquireBillHtml(clWSINT, clBkGso.GetTot( ), *pProfile,
(EPCLCurrency) clReg.m_eDefaultCurrency); #else //Pay Page alone requires a total
pszTot = clWSINT.LookUp("total"); if (!pszTot) return (eRRNoPayTotal); eStat =
AcquireBillHtml(clWSINT, atoi(pszTot), *pProfile, (EPCLCurrency)
clReg.m_eDefaultCurrency); clWSINT << "<INPUT TYPE=HIDDEN NAME=total VALUE = " <<
pszTot << ">.backslash.n"; #endif if (eStat != eSuccess) return (eStat); //error
from db? within AcquireBillHtml clWSINT << "<P>.backslash.n"; // Output Buttons on
Form pszShopUrl = clWSINT.LookUp("shop_url"); if (!pszShopUrl)
PayButtonsHtml(clWSINT; pszRefererUrl, clReg); else PayButtonsHtml(clWSINT,
pszShopUrl, clReg); // Registry Card LookUp clWSINT << "<CENTER><TABLE CELLSPACING =
5><TR><TH>Cards Accepted: </TH>"; RegOpenKeyEx(clReg.m_hStoreKey, "API CDT", 0,
KEY_READ, &hCardsKey); dwlen = sizeof(int); RegQueryValueEx(hCardsKey, "NoOfRows",
0, &dwtype, (LPBYTE)&nNoCards, &dwlen); for (int i = 0; i < nNoCards; i++) {
RegEnumKey(hCardsKey, i, szCardBuf, MAX_PATH + 1); RegOpenKeyEx(hCardsKey,
szCardBuf, 0, KEY_READ, &hCardKey); dwlen = MAX_PATH + 1; retCode =
RegQueryValueEx(hCardKey, "CardPicture", 0, &dwtype, (LPBYTE) szCardPic, &dwlen); if
(retCode != ERROR_SUCCESS) return eRRRegistryFailure; clWSINT << "<TD><IMG SRC = "
<< szCardPic << "></TD>"; RegCloseKey(hCardKey); } RegCloseKey(hCardsKey); clWSINT
<< "</TR></TABLE></CENTER>"; clWSINT << "</FORM>.backslash.n<HR>.backslash.n";
#ifdef_SC // Output static HTML Table clBkGso.ToHtml(clWSINT, NOEDIT); // Output
static Shipping information StaticShipHtml(clWSINT, *pProfile); //Also NO_EDIT
clWSINT << "<HR>.backslash.n"; #else // Pay page alone takes and passes through a
gso if (pszGsoOpaque) clWSINT << pszGsoOpaque << ".backslash.n"; #endif // Rest of
Line from template file if (pszTag) clWSINT << pszTag; } } if (nTagExist != TRUE)
return(eRRNoDynamicTag); else return (eSuccess); } ////////////////////////////////
// Receipt Page
////////////////////////////////////////////////////////////////////////// ///
///////////////#def_SC
//////////////////////////////////////////////////////// // StaticShipHtml
// On Pay page, output Static table of shipping information // based on cookies set
in prior page ////////////////////////////////////////////////////////////////// void
StaticShipHtml(CWSINT& clWSINT, CProf clProfile) { clWSINT << "<CENTER><TABLE
CELLSPACING= 10><CAPTION ALIGN = TOP><B>Ship To<B></CAPTION>.backslash.n"; clWSINT
<< "<TR><TH ALIGN=LEFT>Name</TH><TD>" << clProfile.m_s_name << "</TD>" << "<TH
ALIGN=LEFT>Address Line 1</TH><TD>" << clProfile.m_s_addr1 <<
"</TD></TR>.backslash.n"n; clWSINT << "<TR><TH ALIGN=LEFT>Address Line 2</TH><TD>"
<< clProfile.m_s_addr2 << "</TD>" << "<TH ALIGN=LEFT>City</TH><TD>"
<<clProfile.m_s_city << "</TD></TR>.backslash.n"; clWSINT << "<TR><TH
ALIGN=LEFT>State/Province</TH><TD>" << clProfile.m_s_state << "</TD>" << "<TH
ALIGN=LEFT>Zip/Postal Code</TH><TD>" << clProfile.m_s_zip <<
"</TD></TR>.backslash.n"; clWSINT << "<TR><TH ALIGN=LEFT>Country</TH><TD>" <<
clProfile.m_s_country << "</TD>" << "<TH ALIGN=LEFT>Shipping Method</TH><TD>" <<
clProfile.m_s_ship << "</TD></TR>.backslash.n"; clWSINT << "</TABLE></CENTER><P>"; }
#endif ////////////////////////////////////////////////////////////////// //
StaticBillHtml // On Receipt page, output static table of billing information
////////////////////////////////////////////////////////////// void
StaticBillHtml(CWSINT& clWSINT, CProf clProfile) { /*<TR>Payment
Type</TH>.backslash.n<TD>" <<clProfile.m_b_instrument << "</TD>*/ clWSINT <<
```

```
"<CENTER><TABLE CELLSPACING=10><CAPTION ALIGN = TOP><B>Bill
To<B></CAPTION>.backslash.n"; clWSINT << "<TR ALIGN=LEFT><TH>Account Number</TH><TD
COLSPAN=3>" << clProfile.m_b_card << "</TD><TR>.backslash.n"; clWSINT << "<TR
ALIGN=LEFT><TH>Name on Card</TH><TD>" << clProfile.m_b_name <<
"</TD><TD><B>Expires:</B><I>Month</I> " << clProfile.m_b_expire_month << "
<I>Year</I> " << clProfile.m_b_expire_year << "</TD></TR>.backslash.n"; clWSINT <<
"<TR ALIGN=LEFT><TH>Address Line 1</TH><TD COLSPAN=3>" << clProfile.m_b_addr1 <<
"</TD></TR>.backslash.n"; clWSINT << "<TR ALIGN=LEFT><TH>Address Line 2</TH><TD
COLSPAN=3>" << clProfile.m_b_addr2 << "</TD></TR>.backslash.n"; clWSINT << "<TR
ALIGN=LEFT><TH>City</TH><TD>" << clProfile.m_b_city << "</TD>" << "21
TH>State/Province</TH><TD>" << clProfile.m_b_state << "</TD></TR>.backslash.n";
clWSINT << "<TR ALIGN=LEFT><TH>Country</TH><TD>" << clProfile.m_b_country <<
"</TD><TH>Zip/Postal Code</TH><TD>" << clProfile.m_b_zip <<
"</TD></TR>.backslash.n"; clWSINT << "<TR ALIGN=LEFT><TH>Email</TH><TD>" <<
clProfile.m_b_email << "</TD>" << "<TH>Phone</TH><TD>" << clProfile.m_b_phone <<
"</TD></TR>.backslash.n"; clWSINT <<

Detailed Description Paragraph Table (37):
"</TABLE></CENTER><P>.backslash.n"; }
////////////////////////////////////////////////////////////// // vPOSReceipt //
Generates a receipt from the return block and profile info.
////////////////////////////////////////////////////////////// #ifdef vPOS_OLE
#ifdef_SC void vPOSReceipt(CWSINT& clWSINT, /* CVPCLFinCCTrans */
CVPCL_OleCCAuthOnly *pTxn, CProf& clProfile, CRRReg& clReg, CPOLBk& clBkGso) { #else
void vPOSReceipt(CWSINT& clWSINT, /* CVPCLFinCCTrans */ CVPCL_OleCCAuthOnly *pTxn,
CProf& clProfile, CRRReg& clReg) { #endif #else #ifdef_SC void vPOSReceipt(CWSINT&
clWSINT, CVPCLFinCCTrans *pTxn, CProf& clProfile, CRRReg& clReg, CPOLBk& clBkGso) {
#else void vPOSReceipt(CWSINT& clWSINT, CVPCLFinCCTrans *pTxn, CProf& clProfile,
CRRReg& clReg) { #endif #endif // Set Long cookies (if applicable) struct tm *tmNow;
char szDate[32]; //what is the max date? in this format/ bug time_t tNow
time(&tNow); tNow += clReg.m_nProfileLife * 86400;//ini constant for length of
cookie stay tmNow = localtime(&tNow); strftime(szDate, (size_t)31, "%a, %d-%b-%y
%H:%M:%S GMT", tmNow); if (clWSINT.LookUp("long_cookies")) clWSINT << "Set-Cookie:
cust_profile=" << clProfile.GetCookieLine( ) <<"; expires=" << szDate <<";
path=/.backslash.n"; //Profile cookies #ifdef_SC // Shopping cart sets local cookies
on receipt clWSINT << "Set-Cookie: profile=" << clProfile.GetCookieLine( ) <<
";expires=" << szDate << "; path=/.backslash.n"; //Profile cookies #endif /*clWSINT
<< "; domain = " << clWSINT.LookUp("Server Name") << ";.backslash.n";*/ #ifdef_SC //
Delete shopping basket clBkGso.ToCookies(clWSINT, EXPIRE); #endif clWSINT <<
"Pragma: no-cache.backslash.n"; clWSINT << "Content-type:
text/html.backslash.n.backslash.n"; clWSINT << "<HTML><BODY" <<
clReg.m_szBackgroundString<< ">.backslash.n"; clWSINT << "<A
NAME=jump></A>.backslash.n"; clWSINT << "<CENTER><IMG SRC=" <<
clReg.m_szReceiptBanner << "></CENTER>.backslash.n"; clWSINT << "<CENTER><H2>This is
your receipt. Please save it using the <I>Save As</I> option from the <I>File
Menu</I> in your browser</H2></CENTER>"; //vPOS Return Block char
szGso[PURCH_ORDER_NUM_SZ + 1]; char szTransAmt[AMT_SZ + 1]; char
szDisplayTransAmt[FORMATTED_CURRENCY + 1]; //Extra point for decimal enum
EPCLCurrency eCurr;// = (EPCLCurrency) clReg.m_eDefaultCurrency; enum EPCLDecimals
eDec;// = eTwoDecDigits; char szTime[TRANS_TIME_SZ + 1]; char szPan[ACCT_NUM_SZ +1];
char szExpDate[EXP DATE_SZ + 1]; char szRetRefNum[RET_REF_NUM_SZ + 1];
pTxn->GetRespTransAmt(szTransAmt, AMT_SZ + 1, &eCurr, &eDec);
pTxn->GetPurchOrderNum(szGso, PURCH_ORDER_NUM_SZ + 1);
pTxn->GetRespTransDate(szDate, TRANS_DATE_SZ + 1); pTxn->GetRespTransTime(szTime,
TRANS_TIME_SZ + 1); pTxn->GetRetRefNum(szRetRefNum, RET_REF_NUM_SZ + 1);
pTxn->GetPAN(szPan ACCT_NUM_SZ +1); pTxn->GetExpDate(szExpDate, EXP_DATE_SZ + 1);
clWSINT <<" <CENTER><TABLE BORDER=0 CELLSPACING=10><CAPTION><B>" <<
clReg.m_szShopName << " - Order Number</B> - " << szGso <<
"</CAPTION>.backslash.n<TR ALIGN=LEFT><TH>Time</TH><TD>" << szTime[0] << szTime[1]
<< ":" << szTime[2] << szTime[3] << ":" << &szTime[4] << "</TD><TH>Date</TH><TD>"
szDate[0] << szDate[1] << "/" << szDate[2] << szDate[3] << "/" << szDate[4] <<
"</TD></TR>" << "<TR ALIGN=LEFT><TH>Account Number</TH><TD COLSPAN=3><B>" << szPan
<< "</B></TD></TD>" << "<TR ALIGN=LEFT><TH>Authorization Code</TH><TD>" << "No
Auth?" << "</TD><TH>Reference Number</TH><TD>" << szRetRefNum << "</TD></TR>" <<
"</TABLE></CENTER>"; CurrFormat(atoi(szTransAmt), eCurr, szDisplayTransAmt); clWSINT
<< "<CENTER><FONT SIZE=5>Total = << szDisplayTransAmt <<
"</FONT></CENTER><HR>.backslash.n"; //transtype, time, date, acct #, expire, vPOS
id, transaction type, auth code, ref#, amount // Soft goods fulfillment char
*pszSoftUrl = clWSINT.LookUp("soft_url"); if (pszSoftUrl) clWSINT << pszSoftUrl <<
```

```
"<HR>"; #ifdef_SC // Static Gso, placeholder crap until do LnGrp
clBkGso.ToHtml(clWSINT, NOEDIT); clWSINT << "<HR>"; // Static Billing
StaticBillHtml(clWSINT; clProfile); clWSINT << "<HR>"; // Static Shipping
StaticShipHtml(clWSINT, clProfile); clWSINT << "<HR>"; #else // Static passed gso if
it exists char *pszGso = clWSINT. LookUp("gso_opaque"); if (pszGso) clWSINT <<
pszGso; // Static Billing StaticBillHtml(clWSINT, clProfile); clWSINT << "<HR>";
#endif // Merchant Signature Block (if/when applicable) // Buttons char *pszHomeUrl
= clWSINT.LookUp("home_url"); char *pszShopUrl = clWSINT. LookUp("shop_url");
clWSINT << "<CENTER>.backslash.n<A HREF = " << pszShopUrl << "> <IMG SRC=" <<
clReg.m_szReturnShop << " BORDER = 0></A>.backslash.n" << "<A HREF = << pszHomeUrl
<< "> <IMG SRC=" << clReg.m_szHome << " BORDER = 0></A>.backslash.n" <<
"</CENTER><HR>.backslash.n"; //Acquirer Banner char szPANLo[ACCT_NUM_SZ + 1],
szPANHi[ACCT_NUM_SZ + 1], szBuf[MAX_PATH + 1]; char.backslash.
szTrunCPAN[ACCT_NUM_SZ+1]; HKEY hCardsKey, hCardKey; DWORD dwtype, dwlen; int
nNoCards, nPANLen; long retCode; RegOpenKeyEx(clReg.m_hStoreKey, "API CDT", 0,
KEY_READ, &hCardsKey); dwlen = sizeof(int); RegQueryValueEx(hCardsKey, "NoOfRows",
0, &dwtype, (LPBYTE)&nNoCards, &dwlen); for (int i = 0; i < nNoCards; i++) {
RegEnumKey(hCardsKey, i, szBuf, MAX_PATH + 1); RegOpenKeyEx(hCardsKey, szBuf, 0,
KEY_READ, &hCardKey); dwlen = ACCT_NUM_SZ + 1; retCode = RegQueryValueEx(hCardKey,
"PANLo", 0, &dwtype, (LPBYTE)szPANLo, &dwlen); if (retCode != ERROR_SUCCESS) return;
dwlen = ACCT_NUM_SZ + 1; retCode = RegQueryValueEx(hCardKey, "PANHi", 0, &dwtype,
(LPBYTE)szPANHi, &dwlen); if (retCode != ERROR_SUCCESS) return; nPANLen =
strlen(szPANLo); strncpy(szTruncPAN, szPan, nPANLen); szTruncPAN[nPANLen] =
`.backslash.0`; if((atoi(szTruncPAN) >= atoi(szPANLo)) && (atoi(szTruncPAN) <=
atoi(sZPANHi))) { char szAcquirer[MAX_PATH + 1], szAcquirerBanner[MAX_PATH + 1];
szAcquirer[0] = NULL; szAcquirerBanner[0] = NULL; HKEY hAcquirersKey, hAcquirerKey;
int nNoAcquirers = 0; dwlen = MAX_PATH + 1; RegQueryValueEx(hcardKey, "Acquirer", 0,
&dwtype, (LPBYTE) szAcquirer, &dwlen); RegOpenKeyEx(clReg.m_hStoreKey, "API ADT", 0,
KEY_READ, &hAcquirersKey); dwlen = sizeof(int); retCode =
RegQueryValueEx(hAcquirersKey, "NoOfRows", 0, &dwtype, (LPBYTE)&nNoAcquirers,
&dwlen); for (int j = 0; j < nNoAcquirers; j++) { retCode =
RegEnumKey(hAcquirersKey, j, szBuf, MAX_PATH + 1); // Get jth Acquirer subkey in
szbuf if (retCode != ERROR_SUCCESS) break; if (!strcmp(szBuf, szAcquirer)) {
RegOpenKeyEx(hAcquirersKey, szBuf, 0, KEY_READ, &hAcquirerKey); dwlen = MAX_PATH +
1; retCode = RegQueryValueEx(hAcquirerKey, "AcquirerBanner", 0, &dwtype, (LPBYTE)
szAcquirerBanner, &dwlen); if (retCode != ERROR_SUCCESS) break; clWSINT <<
"<CENTER><IMG SRC=" << szAcquirerBanner << "></CENTER >.backslash.n";
RegCloseKey(hAcquirerKey); break; } } RegCloseKey(hAcquirersKey); break; }
RegCloseKey(hCardKey); } RegCloseKey(hCardsKey); clWSINT << "</HTML>"; }
////////////////////////////////////////////////////////////// //vPOSPay // Create
a PO object and invoke the vPOS
////////////////////////////////////////////////////////////// EStatus
vPOSPay(CWSINT& clWSINT, CRRReg& clReg) EStatus eStat; EPCLTransType eTxn; char
*pszTxn = clWSINT.LookUp("transaction"); char szBuf[MAX_CGI_VAR + 1]; // used for
cgi variable tstore and for number later #ifdef_SC CPOLBk clBkGso; //GSO data
structure #else //Total for transaction char *pszTotal = clWSINT.Lookup("total"); if
(!pszTotal) return(eRRNoPayTotal); #endif //Profile object CProf *pProfile; pProfile
= new CProf( ); if (!pProfile) return (eRRNewFailed); eStat =
pProfile->Init(clWSINT); if (eStat != eSuccess) return (eStat); // Check billing
information if (!(pProfile->m_b_name[0] && pProfile->m_b_addr1[0] &&
pProfile->m_b_city[0] && pProfile->m_b_state[0] && pProfile->m_b_zip[0] &&
pProfile->m_b_country[0] && pProfile->m_b_phone[0] && pProfile->m_b_card[0] &&

Detailed Description Paragraph Table (38):
pProfile->m_b_expire_month[0] && pProfile- >m_b_expire_year[0])) { eStat =
DisplayPayPage(clWSINT, clReg, TRUE); return eStat; } // Payment transaction for a
credit card #ifdef vPOS_OLE CVPCL_OleCCAuthOnly *pTxn; #else CVPCLFinCCTrans *pTxn;
#endif if (pszTxn) { eTxn = eNumTransTypes; if (!strcmp("authonly", pszTxn)) eTxn =
eTransAuthOnly; /* if (!strcmp("authcapture", pszTxn)) eTxn = eTransAuthCapture; if
(!strcmp("offlineauth", pszTxn)) eTxn = eTransOfflineAuth;*/ } else eTxn =
clReg.m_eDefaultAuthTrans; // Create Transaction object switch (eTxn) { case
eTransAuthOnly: #ifdef vPOS_OLE pTxn = new CVPCL_OleCCAuthOnly( ); #else pTxn = new
CVPCL_CCAuthOnly( ); #endif if (!pTxn) return eFailure; // Transaction Init Failure
break; default: return eRRIllegalTransaction; } //Transaction Initialize char
*pszMerchant = clWSINT.Lookup("store"); sprintf(szBuf, "MerchName="); strncat(szBuf,
pszMerchant, (MAX_CGI_VAR-10)); //The 10 is for MerchName= // Connect to the OLE
Automation Server #ifdef vPOS_OLE eStat = pTxn->CreateDispatch( ); if (eSuccess !=
eStat) { return eFailure; } #endif eStat = pTxn->InitTrans(szBuf); if (eStat !=
eSuccess) return eFailure; //eRRTxnInitFailed //GSO Number char* b_gso_num =
```

clWSINT.LookUp("b_gso_num"); if (!b_gso_num) return (eRRNoGsoNum); //Compose Gso object //CPOLPO clPO(&b_gso_num); //Creates shopping basket from CGI/Cookies. This information is borrowed by //Line Group class. For each item in the basket, put it in the PO object. We use a member function //That others using the library cannot use because they may not have a basket object at their disposal. //Those others must use the Set methods directly //Then get prices from database. If prices differ, error code #ifdef_SC eStat = clBkGso.Init(clWSINT, *pProfile, clReg); if (eStat != eSuccess) return (estat); // eStat = clPO.InitFromBk(clBkGso); if (eStat != eSuccess) return (eStat); #endif //set all stuff from profile object //set custcookie //set cust id //set personal message //Pay Page standalone. Call an integrator function, execute vPOS stuff, call an ending function. //The calls before and after are for the integrator to reconcile his database with the vPOS. //GSO VERIFICATION suggestions //Check to see if this purchase order exists in the database & if it is linked properly with this price //Insert GSO and line items into db with before vPOS Txn status //eStat = GsoVerify(b_gso_num, pszTotal); //For integrator to fill in. //if (eStat != eSuccess) return eStat; //Failed lookup check #ifdef _SC int nTot; /* nTot = clBkGso.GetTot( ) * 100; if (((clBkGso.GetTot( ) * 100) - nTot) >= .5) ++nTot; sprintf(szBuf "%.2f", nTot/100.0) ; //Transaction Amount, hack to get past 2 digits*/ //erase szBuf below. Lose precision by flooring this integer. need to define round up/down sprintf(szBuf, "%d", (int)clBkGso.GetTot( )); pTxn->SetReqTransAmt(szBuf, (EpCLCurrency) clReg.m_eDefaultCurrency, eTwoDecDigits); #else //Amount NumClean(pszTotal); pTxn->SetReqTransAmt(pszTotal, (EpCLCurrency) clReg.m_eDefaultCurrency, eTwoDecDigits); #endif //GSO Num pTxn->SetPurchOrderNum(b_gso_num); //Retry Counter pTxn->SetRRPid(1); //The first time a transaction is executed this must be set to 1 //AVS Data if (clReg.m_VS) { char avs_zip[ZIP_SZ + 1]; strncpy(avs_zip, pProfile->m_b_zip, ZIP_SZ); avs_zip[ZIP_SZ] = NULL; NumClean(avs_zip); pTxn->SetAVSData(avs_zip); } pTxn->SetBName(pProfile->m_b_name); pTxn->SetBStreetAddress1(pProfile->m_b_addr1); pTxn->SetBStreetAddress2(pProfile->m_b_addr2); pTxn->SetBCity(pProfile->m_b_city); pTxn->SetBStateProvince (pProfile->m_b_state); pTxn->SetBZipPostalCode(pProfile->m_b_zip); //Insert as is zip into db pTxn->SetBCountry(pProfile->m_b_country); pTxn->SetBEMail(pProfile->m_b_email); pTxn->SetBDayTimePhone(pProfile->m_b_phone); //Card Number and expiry date NumClean(pProfile->m_b_card); char szDate[DB_MONTH_SZ + DB_YEAR_SZ + 1]; strncpy(szDate, pProfile->m_b_expire_month DB_MONTH_SZ); szDate[DB_MONTH_SZ] = NULL; strncat(szDate, pProfile->m_b_expire_year, DB_YEAR_SZ); pTxn->SetPAN(pprofile->m_b_card); pTxn->SetExpDate(szDate); //Execute Transaction eStat = pTxn->ExecuteTrans( ); if (eStat != eSuccess) return eStat; //DB or Internal Error of some kind //Transaction Shutdown eStat = pTxn->ShutDownTrans( ); if (eStat != eSuccess) return eFailure; //eRRTxnShutFailed //Gso affer for integrator to fill in //Gso reconcile(success or failure, gso_number); //Delete cookies GSO. Set shipping/billing cookies. Send receipt - member function of PO object #ifdef_SC vPOSReceipt(clWSINT, pTxn, *pProfile, clReg, clBkGso); //This should be PO object #else vPOSReceipt(clWSINT, pTxn, *pProfile, clReg); //Use Get Methods for Receipt #endif #ifdef vPOS_OLE // Disconnect from the server pTxn->ReleaseDispatch( ); #endif return (eSuccess);

Other Reference Publication (18):
Gosling, et al., The Java Language Environment a White Paper, Sun Microsystems Computer Company, (May 1995).

Other Reference Publication (78):
AP Online, AP Financial News At 9:10 a.m. EST Thursday, Feb. 1, 1996, Feb. 1, 1996, pp. 118-122.

Other Reference Publication (289):
Newsbytes News Network, France--Bull Forms Smart Card Subsidiary Apr. 13, 1995, Apr. 13, 1995, pp. 274-275.

Other Reference Publication (322):
PC Week, IBM Takes Charge of E-Commerce: Plans Client, Server Apps Based on SET; NetCommerce Electronic Commerce System; Product Announcement, Apr. 29, 1996, pp. 118-119.